Framework

Spring Boot

[1. 引言 1-2](#_Toc11168605)

[1.1. Spring 传统项目整合弊病 1-2](#_Toc11168606)

[1.2. Spring 家族主要成员构成 1-2](#_Toc11168607)

[1.3. Spring 应用中的技术趋势 1-3](#_Toc11168608)

[2. Spring Boot 简介 2-3](#_Toc11168609)

[2.1. Spring Boot概述 2-3](#_Toc11168610)

[2.2. Spring boot入门案例 2-4](#_Toc11168611)

[2.2.1. 初始环境配置 2-4](#_Toc11168612)

[2.2.2. 项目创建及结构分析 2-5](#_Toc11168613)

[2.2.3. 项目业务基本实现及测试 2-8](#_Toc11168614)

[2.2.4. Runner启动器应用分析及实践 2-9](#_Toc11168615)

[3. Spring Boot 整合数据源 3-10](#_Toc11168616)

[3.1. 单数据源整合实现 3-10](#_Toc11168617)

[3.1.1. 使用默认HikariCP 3-10](#_Toc11168618)

[3.1.2. 整合Druid连接池 3-13](#_Toc11168619)

[3.2. 多数据源整合实现（扩展） 3-14](#_Toc11168620)

[4. Spring Boot 整合ORM框架 4-14](#_Toc11168621)

[4.1. 整合mybatis框架 4-14](#_Toc11168622)

[4.1.1. 添加mybatis启动依赖 4-14](#_Toc11168623)

[4.1.2. 配置mybatis框架并测试 4-15](#_Toc11168624)

[4.1.3. 基本业务实现及单元测试 4-15](#_Toc11168625)

[5. Spring Boot整合Web资源 5-16](#_Toc11168626)

[5.1. Web资源整合入门 5-16](#_Toc11168627)

[5.1.1. 添加项目依赖 5-16](#_Toc11168628)

[5.1.2. 项目业务编写测试 5-17](#_Toc11168629)

[5.2. Web资源整合进阶 5-19](#_Toc11168630)

[5.2.1. 定义静态页面 5-19](#_Toc11168631)

[5.2.2. 配置映射前缀后缀 5-19](#_Toc11168632)

[5.2.3. 控制层方法定义及测试 5-19](#_Toc11168633)

[5.2.4. 服务器简易配置实现 5-20](#_Toc11168634)

[5.2.5. 服务器资源热部署实现 5-20](#_Toc11168635)

[6. Spring Boot 应用加强 6-20](#_Toc11168636)

[6.1. Lombok插件应用(扩展) 6-20](#_Toc11168637)

[6.2. mybatis-plus插件应用（扩展） 6-22](#_Toc11168638)

[7. 总结 7-23](#_Toc11168639)

[7.1. 重点难点 7-23](#_Toc11168640)

[7.2. FAQ分析 7-23](#_Toc11168641)

[7.3. Bug分析 7-23](#_Toc11168642)

# 引言

## Spring 传统项目整合弊病

在传统的spring应用项目中：

1. 配置相对复杂（例如大量xml）
2. 依赖冲突严重（maven依赖版本不同）
3. 扩展相对复杂

思考：如何让初级程序员从复杂的配置中解脱出来呢？

## Spring 家族主要成员构成

Spring诞生于2002年，成型于2003，目前已经发展到了Spring5.x版本，支持JDK-11及JavaEE8。现在的spring已经不仅仅是framework，还扩展了spring boot，spring cloud等。

## Spring 应用中的技术趋势

Spring 自5.x以后：

1. 只能向下兼容到JDK8，并支持Kotlin语言。
2. WebFlux异步编程技术突起，但全面落地还需要一些时日。
3. Portlet过时了，Velocity也不在维护，JasperReport不再流行等。
4. Spring boot，Spring cloud 异军突起。

说明:学技术一定要了解技术发展史，并对技术的发展有一定的前瞻性。

# Spring Boot 简介

## Spring Boot概述

现在软件市场已经形成一定的规模，系统从单体架构，分布式架构，再到微服务架构，其整个架构体系正在发生很大变化，企业现在更注重技术的开箱即用，更注重技术在生态圈中的深度融合，更注重轻量级的运维。由此spring boot诞生。

Spring boot构建与Spring 框架基础之上，基于快速构建理念，提供了自动配置功能，可实现其开箱即用特性，可以简化整个项目的配置。其核心主要包括如下几个方面：

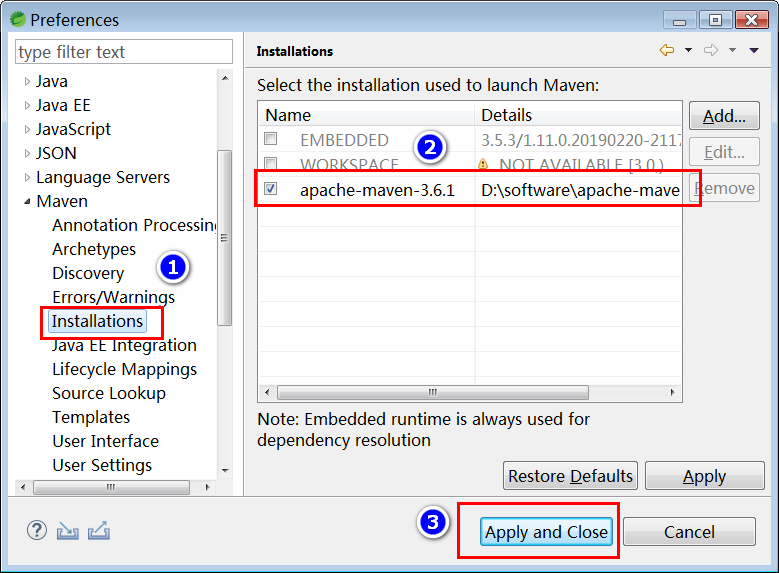
1. 自动配置(Auto Configuration)
2. 起步依赖(Starter Dependency)
3. 健康检查(Actator)
4. …

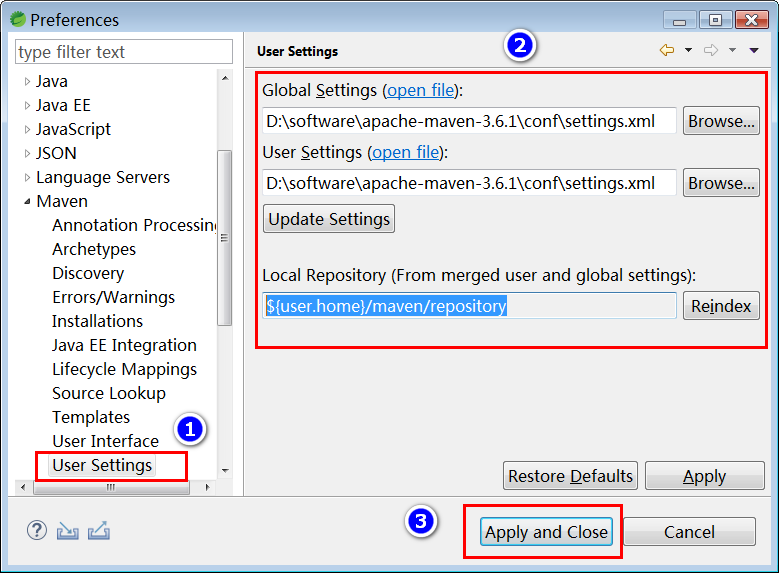
说明：Spring Boot地址：<https://spring.io/projects/spring-boot>

## Spring boot入门案例

### 初始环境配置

1. 下载JDK1.8，并进行环境变量配置
2. 下载maven(apache-maven-3.6.1)，并进行相关配置(本地库，私服)。
3. 下载sts最新版(sts-4.2.2.RELEASE)，并进行maven配置.

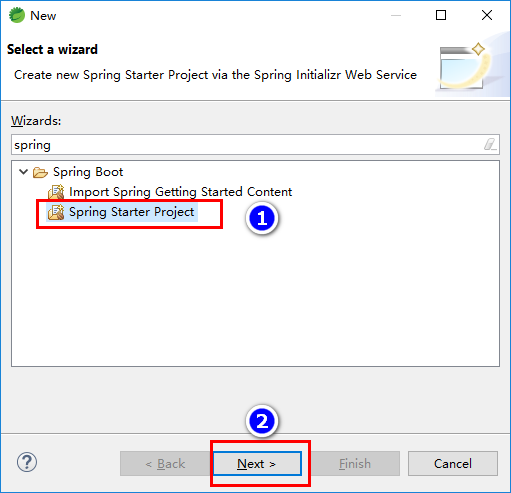




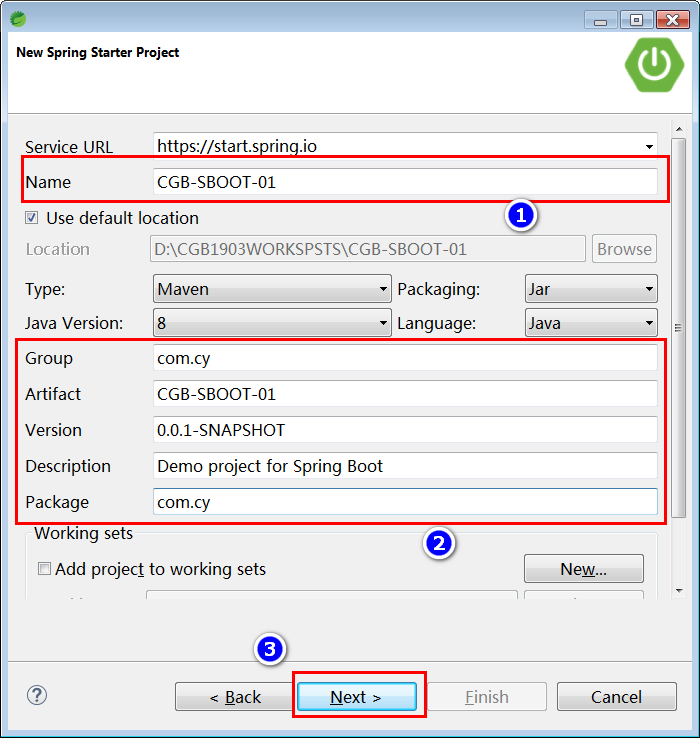
### 项目创建及结构分析

打开STS 集成开发工具，并基于如下步骤实现项目创建。

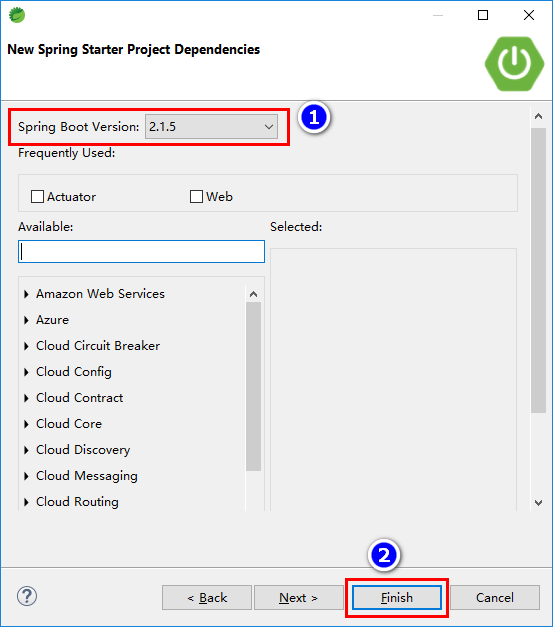
Step01:打开项目新建窗口（快捷键 ctrl+n）, 搜索spring，选择spring starter 项目。



Step02 填写项目基本信息。



Step03 选择Spring Boot版本。



### 项目业务基本实现及测试

业务描述：创建一个cache类然后交给spring 管理。

**package** com.cy.pj.common.cache;

@Component

@Scope("singleton")

**public** **class** Cache {

**public** Cache() {

System.***out***.println("cache()");

}

@PostConstruct

**public** **void** init() {

System.***out***.println("init()");

}

@PreDestroy

**public** **void** destory() {

System.***out***.println("destory");

}

}

修改sringboot 启动类，进行bean的获取及测试。

@RunWith(SpringRunner.**class**)

@SpringBootTest

**public** **class** CgbSboot01ApplicationTests {

@Autowired

**private** ApplicationContext ctx;

@Test

**public** **void** testCache() {

Cache cache=

ctx.getBean("cache", Cache.**class**);

System.out.println(cache);

}

}

### Runner启动器应用分析及实践

如果你想在Spring Boot启动的时候运行一些特定的代码，你可以实现接口 ApplicationRunner或者 CommandLineRunner，这两个接口实现方式一样，它们都只提供了一个run方法，其中：

1. **CommandLineRunner**：启动获取命令行参数。
2. **ApplicationRunner**：启动获取应用启动的时候参数。

SpringBoot在项目启动后会遍历所有实现CommandLineRunner的类并执行run方法。

例如：

@SpringBootApplication

**public** **class** CgbSpringbootApplication **implements** CommandLineRunner{

@Autowired

**private** ApplicationContext ctx;

@Override

**public** **void** run(String... args) **throws** Exception {

System.***out***.println(ctx.getBean("cache"));

}

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(CgbSpringbootApplication.**class**, args);

}

}

说明：

如果启动的时候有多个ApplicationRunner和CommandLineRunner，想控制它们的启动顺序，可以实现 org.springframework.core.Ordered接口或者使用 org.springframework.core.annotation.Order注解。

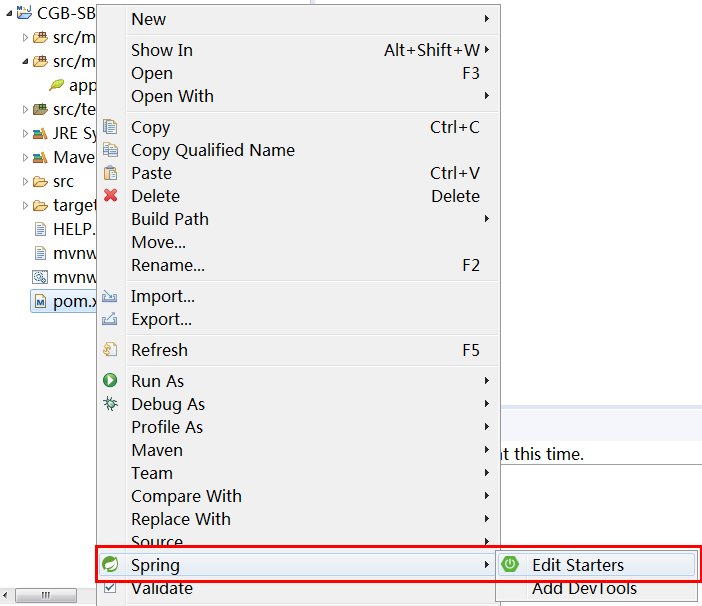
# Spring Boot 整合数据源

## 单数据源整合实现

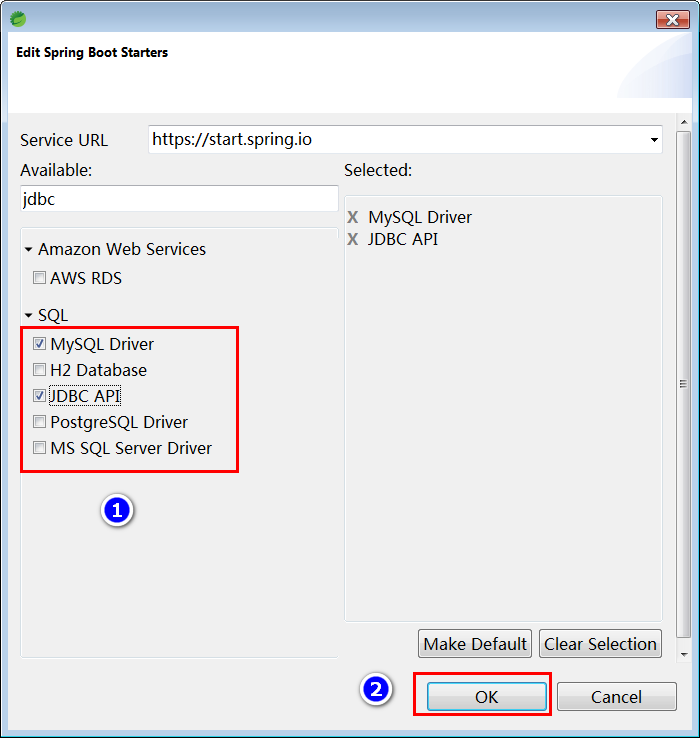
### 使用默认HikariCP

Step01:添加依赖

编辑项目中pom.xml,右键项目的pom.xml文件,选择spring,如下图所示:



基于基本业务添加如下依赖：



依赖添加以后,在pom.xml文件中会自动添加如下两个依赖配置

mysql数据库驱动依赖

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

spring对象jdbc支持

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

Step02连接池简易配置:

打开application.properties配置文件，添加如下内容。

spring.datasource.url=jdbc:mysql:///jtsys?useSSL=false&serverTimezone=GMT%2B8

spring.datasource.username=root

spring.datasource.password=root

Step03:单元测试

@RunWith(SpringRunner.**class**)

@SpringBootTest

**public** **class** CgbSbootApplicationTests {

@Autowired

**private** DataSource dataSource;

@Test

**public** **void** testDataSource() **throws** Exception{

System.out.println(dataSource.getConnection());

}

}

### 整合Druid连接池

Step01:添加druid启动依赖

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid-spring-boot-starter</artifactId>

<version>1.1.13</version>

</dependency>

Step02:配置数据源

打开application.properties配置文件，修改连接池内容配置。

spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

spring.datasource.url=jdbc:mysql:///jtsys?useUnicode=true&characterEncoding=utf-8&serverTimezone=GMT

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

简化写法

spring.datasource.url=jdbc:mysql:///jtsys?useUnicode=true&characterEncoding=utf-8&serverTimezone=GMT

spring.datasource.username=root

spring.datasource.password=root

Step03:单元测试

@RunWith(SpringRunner.**class**)

@SpringBootTest

**public** **class** CgbSbootApplicationTests {

@Autowired

**private** DataSource dataSource;

@Test

**public** **void** testDataSource() **throws** Exception{

System.out.println(dataSource.getConnection());

}

}

## 多数据源整合实现（扩展）

自己课后扩展(作业)

# Spring Boot 整合ORM框架

## 整合mybatis框架

### 添加mybatis启动依赖

参考官网 mybatis.org/spring

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>2.0.1</version>

</dependency>

### 配置mybatis框架并测试

在mybatis的配置文件application.properties文件中添加如下内容

mybatis.configuration.default-statement-timeout=30

mybatis.configuration.map-underscore-to-camel-case=true

### 基本业务实现及单元测试

定义业务接口

**package** com.cy.pj.dao;

**import** org.apache.ibatis.annotations.Delete;

**import** org.apache.ibatis.annotations.Mapper;

@Mapper //3.4

**public** **interface** SysLogDao {

@Delete("delete from sys\_logs where id=#{id}")

**int** deleteObject(Integer id);

}

定义测试类

@RunWith(SpringRunner.**class**)

@SpringBootTest

**public** **class** SysLogDaoTests {

@Autowired

**private** SysLogDao sysLogDao;

@Test

**public** **void** testSysLogDao() {

**int** rows=sysLogDao.deleteObject(10);

System.out.println("rows="+rows);

}

}

### 拓展业务实现及单元测试

Step01:在SysLogDao中定义添加如下方法:

**int** deleteObjects(

@Param("ids")Integer... ids);

说明:当接口方法对应的映射语句相对比较复杂时,建议将映射语句写到对应映射文件

Step02:在src/main/resources目录下创建mapper/sys目录,然后在其目录添加

SysLogMapper.xml,并添加如下内容:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace=*"com.cy.pj.sys.dao.SysLogDao"*>

<delete id=*"deleteObjects"*>

delete from sys\_logs

where id in <!-- (1,2,3,4,5) -->

<foreach collection=*"ids"*

open=*"("*

close=*")"*

separator=*","*

item=*"id"*>

#{id}

</foreach>

</delete>

</mapper>

Step03:在application.properties文件中添加如下配置

mybatis.mapper-locations=classpath:/mapper/sys/\*.xml

Step04:在SysLogDaoTests类中添加如下单元测试方法进行单元测试:

@Test

**public** **void** testDeleteObjects() {

**int** rows=

sysLogDao.deleteObjects(17,18);

System.***out***.println(rows);

}

## 整合mybatis框架进阶

### 日志进阶配置

在spring boot整合mybatis框架时,假如希望在测试运行时输出sql信息,则需要进行日志的相关配置,例如打开application.properties文件然后添加如下日志级别配置.

logging.level.com.cy=DEBUG

说明:其中level单词后面为字节项目的包结构

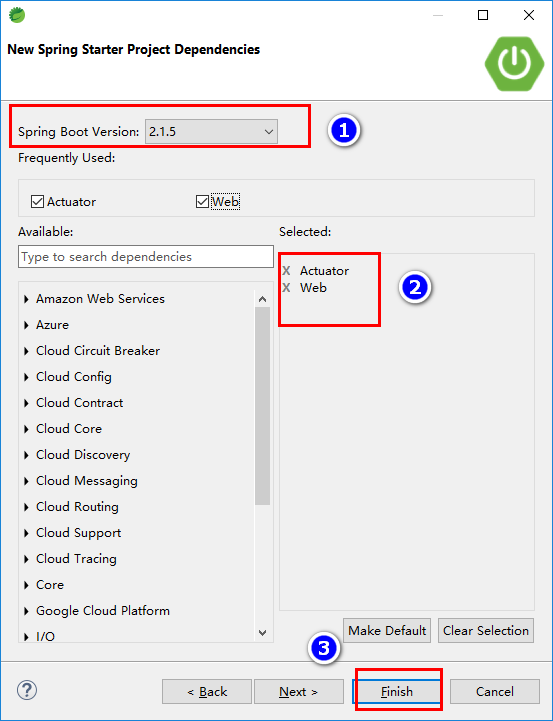
# Spring Boot 整合Web资源

## Web资源整合入门

基于STS创建WEB应用项目（打包方式选择jar包方式），实现简易Spring Boot项目访问。

### 添加项目依赖

Step01:编辑pom.xml文件,添加web和actuator依赖。其中actuator在spring项目中主要负责健康检查以及一些监控功能。



### 项目业务编写测试

Step01:编写HelloController类并将其交给spring管理。

**package** com.cy.pj.sys.controller;

@Controller

**public** **class** HelloController {

@RequestMapping("doSayHello")

@ResponBody

**public** String doSayHello() {

**return** "hello spring";

}

}

}

启动项目（WEB项目内置tomcat，启动时会自动启动tomcat）

**package** com.cy;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** CgbSpringApplication{

**public** **static** **void** main(String[] args)**throws** Exception {

SpringApplication.*run*(CgbSpringApplication.**class**, args);

}

}

浏览器测试方式：

打开chrome浏览器，在地址栏直接输入如下地址

http://localhost:8080/doSayHello

健康检查分析：在浏览器中输入

<http://localhost:8080/actuator/health>

假如希望查看更多actuator选项，可以在spring boot中配置文件application.properties中添加如下语句:

management.endpoints.web.exposure.include=\*

此时在浏览器地址栏可以输入

<http://localhost:8080/actuator/beans> 查看所有的

spring 容器中的bean信息

## Web资源整合进阶

业务描述,通过spring boot资源整合配置实现页面呈现.

### 定义静态页面

在src/main/resources的templates目录下创建pages目录,并在目录中创建hello.html文件，其内容自己定义。

### 配置映射前缀后缀

添加thymeleaf模板引擎依赖（有此依赖可以去除web依赖）

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

在application.properties文件中添加视图解析器

spring.thymeleaf.prefix=classpath:/templates/pages/

spring.thymeleaf.suffix=.html

### 控制层方法定义及测试

@RequestMapping("doSayHelloUI")

**public** String doSayHelloUI() {

**return** "hello";

}

启动项目，然后在在浏览器输入：

http://localhost:8080/doSayHelloUI

### 服务器简易配置实现

打开application.properties文件,添加如下内容

server.port=80

server.tomcat.max-threads=1000

server.servlet.context-path=/

其中:

1. server.port 表示服务器启动端口
2. server.tomcat.max-threads=1000 表示最大线程数
3. server.servlet.context-path=/ 表示context路径配置

### 服务器资源热部署实现

Springboot web项目修改了某个类以后,默认不会自动重新部署加载,需要手动重启服务器,假如需要自动部署,可以添加如下依赖:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

说明:此依赖可通过STS 工具中的Edit starters方式添加

# Spring Boot 扩展应用加强

## Lombok插件应用(扩展)

愿景分析：能否省略pojo类中set/get/toString方法的编写,能否简化项目中日志对象Logger的创建。

Step01：添加依赖

<dependency>

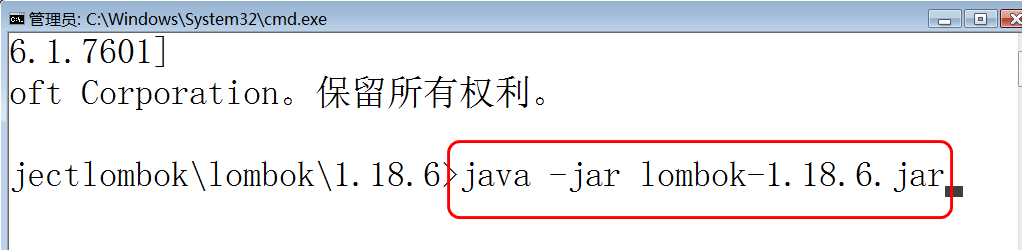
<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

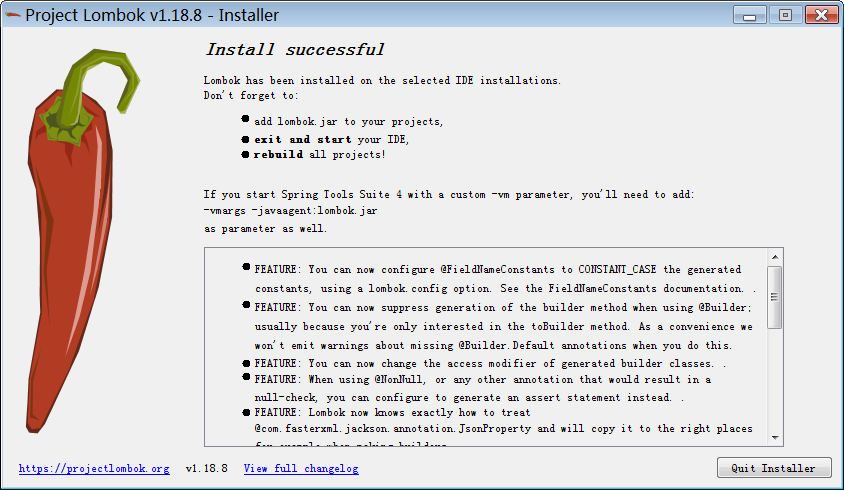
Step02:根据pom.xml文件中的配置.找到lombok文件夹下的lombok-xxx.jar文件的具体位置。

Step03:利用java –jar 指令启动lombok插件，例如



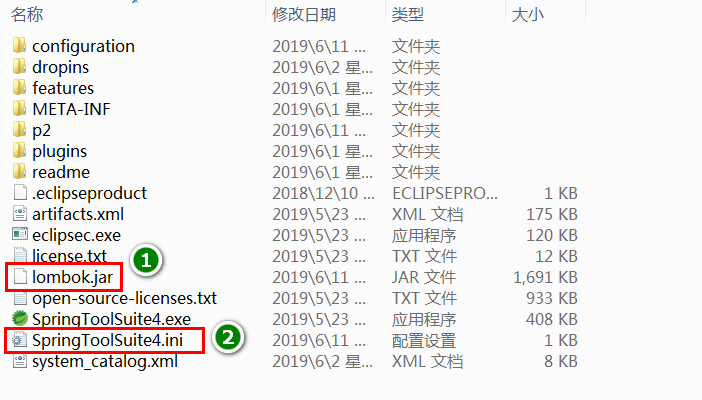
然后在弹出的框中选择sts启动执行文件(SpringToolSuite4.exe)进行安装,

安装成功以后会出现如下弹出框.



安装成功以后可在STS工具的根目录创建一个lombok.jar文件,同时会在SpringToolSuite4.ini文件的尾部添加如下内容:

-javaagent:D:\software\sts-4.2.2.RELEASE\lombok.jar,如下图所示:



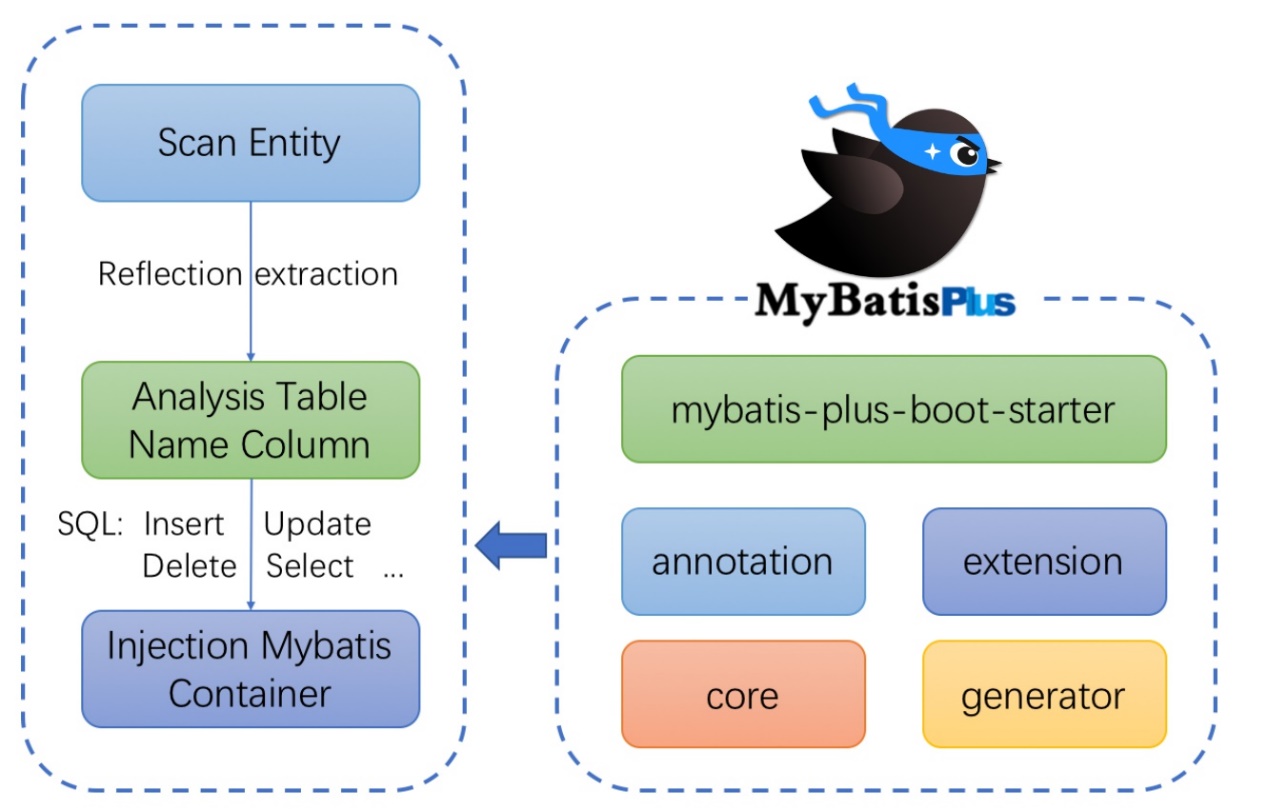
说明:安装成功以后重启sts 工具.(假如安装成功以后再次启动sts,此时无法启动,可能与STS所在目录有关系,可以将安装目录下的lombok.jar文件拷出到特定目录,然后修改SpringToolSuite4.ini文件的最后一行的lombok.jar路径即可)

lombok插件测试使用:

在业务类上添加@SLF4J注解,然后在类的方法内部使用log变量进行日志输出

## mybatis-plus插件应用（扩展）

[MyBatis-Plus](https://github.com/baomidou/mybatis-plus)（简称 MP）是一个 [MyBatis](http://www.mybatis.org/mybatis-3/) 的增强工具，在 MyBatis 的基础上只做增强不做改变，为简化开发、提高效率而生。



# 总结

## 重点难点

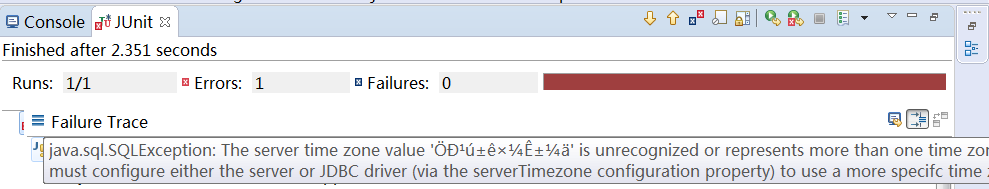
1. Spring boot 核心
2. Spring boot 项目的创建及启动配置
3. Spring boot 整合数据源,mybatis,spring mvc的实现.

## FAQ分析

1. 创建spring starter 项目时pom.xml文件错误.(检查maven配置,网络)
2. 项目中application.properties文件没有提示.(sts版本,文件打开方式)
3. SpringBoot 启动动类所在包(应该为所有自定义bean所在的根包)
4. SpringBoot 内置的连接池是什么(HiKariCP)
5. MySQL 最新驱动程序类?(com.mysql.cj.jdbc.Driver)
6. SpringBoot 连接数据库的配置没有问题,数据库连不上(检查服务启动)
7. SpringBoot整合mybatis时候是否可以同时存在注解和xml的映射
8. SpringBoot整合mybatis时如何实现SQL日志的输出?
9. SpringBoot默认支持热部署吗?(需要添加依赖)

## Bug分析

Bug-01:



解决方案:

1. 降低数据库驱动版本 (传统的maven项目通常会这么做)
2. 在url中追加时区配置(serverTimezone=GMT%2B8)

Bug-02

Caused by: java.net.BindException: Address already in use: bind

at sun.nio.ch.Net.bind0(Native Method) ~[na:1.8.0\_45]

at sun.nio.ch.Net.bind(Unknown Source) ~[na:1.8.0\_45]

at sun.nio.ch.Net.bind(Unknown Source) ~[na:1.8.0\_45]

问题分析:JVM端口可能被占用

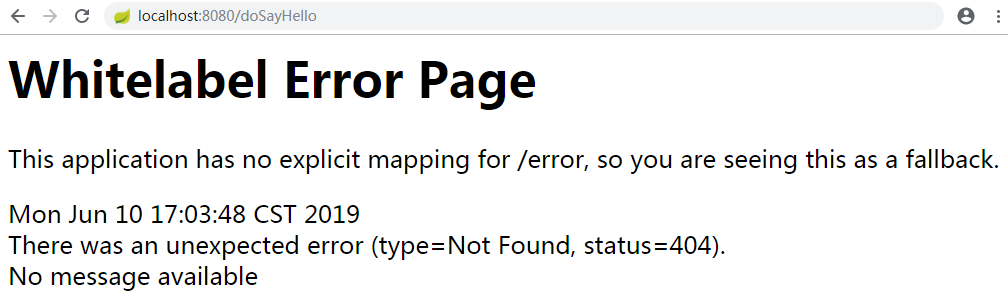
解决方案:

1. 关闭已启动的tomcat,重新启动
2. 杀进程

2.1)查找进程id:netstat -ano|findstr "8080"

2.2)基于进程id杀进程:taskkill /f /pid 进程id

Bug-03

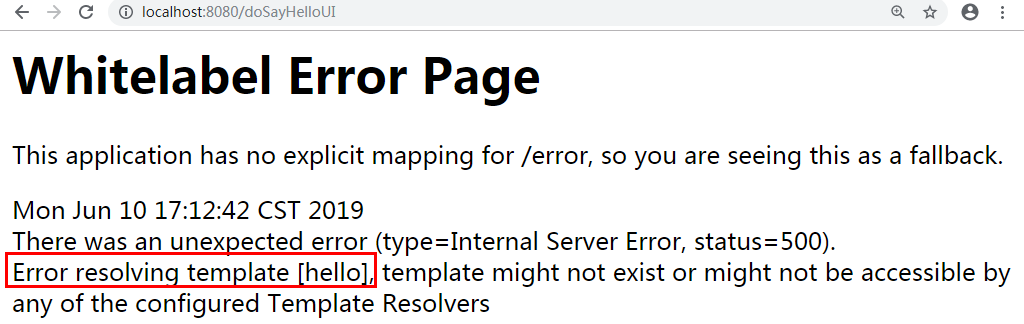


问题描述:客户端请求的服务端资源没找到

解决方案:

1. 检查tomcat启动是否OK
2. 检查url对应的服务端资源是否匹配(客户端url与controller定义url)
3. 检查controller的包结构(所在包应在主启动类所在包的同包或子包中)
4. 检查项目中是否已添加thymeleaf依赖.

BUG-04



问题分析:可能试图解析器的配置有问题

解决方案:检查视图解析器前缀,后缀的配置.

BUG-05

Caused by: java.lang.IllegalArgumentException:

Mapped Statements collection already contains value for

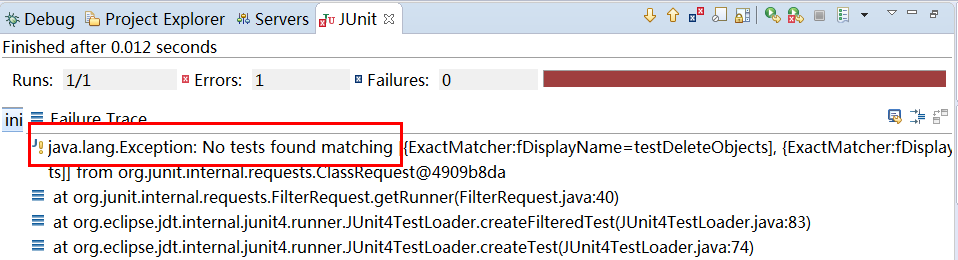
com.cy.pj.sys.dao.SysLogDao.deleteObject. please check file

[D:\CGB1903WORKSPSTS\CGB-SBOOT-02\target\classes\mapper\sys\SysLogMapper.xml]

and com/cy/pj/sys/dao/SysLogDao.java (best guess)

问题分析:可能是SysLogDao对应的映射元素有重复.

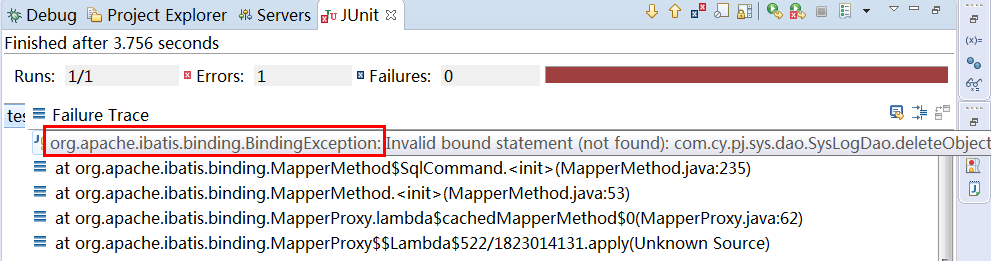
BUG-06



问题分析:

1. 单元测试方法可能使用了private修饰,有返回值或有参数.
2. spring boot项目有多个启动类

BUG-07



问题分析:

1. 接口类全名与映射文件中的命名空间不一致
2. 接口方法名与映射文件中的方法名不一致
3. 接口方法参数与映射文件可接收的参数不匹配
4. 接口对应的映射文件路径配置不正确(mapper-locations)